Document made available under the Patent Cooperation Treaty (PCT)

International application number: PCT/EP05/050500

International filing date: 07 February 2005 (07.02.2005)

Document type: Certified copy of priority document

Document details: Country/Office: DE
Number: 10 2004 007 232.9
Filing date: 13 February 2004 (13.02.2004)

Date of receipt at the International Bureau: 06 April 2005 (06.04.2005)

Remark: Priority document submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b)

PCT/EP2005 / 050500

BUNDESREPUBLIK DEUTSCHLAND

07. 03. 2005

Prioritätsbescheinigung über die Einreichung einer Patentanmeldung

Aktenzeichen:	10 2004 007 232.9
Anmeldetag:	13. Februar 2004
Anmelder/Inhaber:	Siemens Aktiengesellschaft, 80333 München/DE
Bezeichnung:	Rekonfigurierbare Architektur zur parallelen Berechnung beliebiger Algorithmen
IPC:	G 06 F 15/76

Die angehefteten Stücke sind eine richtige und genaue Wiedergabe der ursprünglichen Unterlagen dieser Patentanmeldung.

München, den 25. Februar 2005
Deutsches Patent- und Markenamt
Der Präsident
Im Auftrag

Dzierzon

A 9161
03/00
EDV-L

Beschreibung

Rekonfigurierbare Architektur zur parallelen Berechnung beliebiger Algorithmen

1 Einleitung

Die vorliegende Erfindung bezieht sich auf eine Architektur einer Rechnereinrichtung zur parallelen Berechnung von Algorithmen mit wenigstens einem Schaltnetzwerk.

2 Stand der Technik

2.1 Bekannte Rechnermodelle

Allgemein wird die sogenannte ‚Von-Neumann-Architektur', wie sie aus der prinzipiellen Darstellung der Figur 1 hervorgeht, als Universalrechnereinrichtung angesehen. Darunter ist zu verstehen, dass mithilfe eines Rechners, auf dieser Architektur mit den Komponenten Central Processing Unit [CPU, darin enthalten Control Unit (CU) und Arithmetical-Logical Unit (ALU)], Memory (Speicher), Input/Output (Ein-/Ausgabe) und Bussystem basierend, alle algorithmierbaren Probleme prinzipiell berechnet werden können. Die Einstellung eines solchen Rechners auf das jeweilige Problem erfolgt durch ein Programm, d.h., eine textuelle Beschreibung des Algorithmus z.B. in den Programmiersprachen C, C++ oder Java. Dieses Programm wird durch einen Übersetzer (Compiler), selbst ein Programm darstellend, in ein Maschinen-lesbares Programm übersetzt.

Die Programmbearbeitung erfolgt in den Rechner, die auf der Von-Neumann-Architektur nach Figur 1 und ähnlichen Architekturen (z.B. einer Harvard- oder modifizierten Harvard-Architektur) basieren, prinzipiell auf sequenzielle Weise. Dies ist so zu verstehen, dass der Algorithmus, bestehend aus einer Menge von Maschinen-Instruktionen, dadurch bearbeitet wird, dass die erste Instruktion bekannt ist. Die einem Be-

fehl nachfolgende Instruktion ist entweder die im Speicher an der nächsten Adresse stehende (normaler Programmfluss), oder die letzte ausgeführte Instruktion war ein Sprungbefehl, der den aktuellen Programmstand an eine andere Stelle versetzt. Das interne Register, das den aktuellen Programmstand speichert, wird mit ‚Program Counter' (PC) bezeichnet.

Dieses Prinzip der sequenziellen Bearbeitung bedeutet, dass sich zu einem Zeitpunkt genau eine Instruktion in der Bearbeitung befindet. Es wird als Von-Neumann-Prinzip bezeichnet. Moderne Architekturen, die als RISC (Reduced Instruction-Set Computing), superskalar oder VLIW (Very Long Instruction Word) bezeichnet werden, führen zwar zu einem Zeitpunkt mehr als eine Instruktion aus; das Prinzip der Sequenzialität bleibt jedoch erhalten. Insgesamt wird dieses Ausführungsprinzip als 'zeit-sequenziell' (Computing in Time) bezeichnet, was andeutet, dass der Algorithmus Zeit benötigt.

Ein gänzlich anderes Prinzip der Programmbearbeitung ist in programmierbaren Logikbausteinen [PLDs(Programmable Logic Devices), entsprechend Figur 2] bzw. deren bekanntester Implementierung, den FPGAs (Field-Programmable Gate Arrays), vorgesehen. Auch diese Architektur ist universell, d.h. für jedes algorithmierbare Problem einsetzbar. Die Programmierung erfolgt hierbei so, dass elementare Recheneinheiten, meist auf Bit-Ebene definiert und daher als Logikelemente bezeichnet, in einem Netzwerk verschaltet werden. Diese Form der Programmierung wird meist als ‚Konfiguration' bezeichnet.

Die Programmbearbeitung in einem PLD erfolgt im Unterschied zum Von-Neumann-Prinzip mit maximaler Parallelität. Die gesamte Konfiguration eines Bausteins kann als eine einzige Instruktion - im Gegensatz zum Von-Neumann-Prinzip allerdings nicht fest definiert, sondern zusammengesetzt - aufgefasst werden, die in einem Zyklus komplett bearbeitet wird. Die Zykluszeit, häufig mit einem externen Takt in Verbindung gebracht, ist dann von der Komplexität der Zusammensetzung ab-

hängig. Hierdurch kommt ein im Vergleich zu Prozessoren niedrigerer Takt zum Einsatz, der aber durch die Parallelität der Ausführung mehr als ausgeglichen wird. Dieses Prinzip der Ausführung wird als ‚Computing in Space' bezeichnet.

2.2 Deterministische endliche Automaten

Eines der wesentlichen Modelle zur Algorithmierung von Problemen sind deterministische endliche Automaten [DEAs, auch DFAs (deterministic finite automata)]. Diese werden in der Technik auch als ‚Finite State Machines' (FSMs gemäß der prinzipiellen Darstellung nach Figur 3) bezeichnet. Dieses Modell betrachtet das Problem als eine Folge von Zuständen mit definierten Übergangsfunktionen (Next State Decoder) zwischen diesen, abhängig von den Eingangswerten. Obwohl das Modell des DEA theoretisch nicht so mächtig ist wie das des Von-Neumann-Modells, können in der Praxis beliebige Probleme, ggf. mit Zusatz im Modell, gelöst werden.

Das größte Problem dieser DEAs besteht darin, dass ihre Funktionen mit der Anzahl der Zustände in exponentieller Weise anwachsen, den Flächenbedarf an Halbleitermaterial (insbesondere Silizium) betreffend. Aus diesem Grund wählt man gerne Modelle, die aus vielen, miteinander kommunizierenden Automaten bestehen. Ein derartiges Rechnermodell wird als ‚komplexer, kooperierender Automat' bezeichnet.

2.3 Darstellung Boolescher Funktionen

Eine *Boolesche Funktion* oder *Schaltfunktion* ist eine Abbildung $f: \mathbf{B}^m \rightarrow \mathbf{B}^n$, mit $\mathbf{B}=\{0,1\}$, die in der Form $f = (f_1, f_2, \ldots f_n)$, also als Vektor von Funktionen $f_k: \mathbf{B}^m \rightarrow \mathbf{B}$ dargestellt werden kann. Im Folgenden wird daher nur von Funktionen f_k mit einem Ausgangswert ausgegangen; diese werden allgemein als f bezeichnet.

Es ist weiterhin bekannt, dass f in Form einer disjunktiven oder konjunktiven Normalform darstellbar ist. Für die disjunktive Normalform bedeutet dies, dass
f = z1 + z2 … + zk, mit k = 1, 2, ..., 2m und '+' als OR-Operator (logisches ODER) (1)
und
zi = y1 * y2 * … yn mit i = 1, 2, …, k mit '*' als AND-Operator (logisches UND) (2)
gilt. Es werden natürlich nur solche zi verwendet, für die die zu beschreibende Funktion den Wert TRUE oder '1' erhält. Die Bezeichnung yj bestimmt dabei, wie ein Inputbit i_k aus dem Inputvektor x = (i_1, i_2, …i_k) werden soll. Während für die Inputbits nur Werte aus der Menge {0, 1} zugelassen sind, muss dies für yj geändert werden: Diesen Werten wird eines aus den drei Attributen {0, 1, -} zugewiesen. Das Attribut '1' für yj bedeutet dabei, dass i_k unverändert genommen wird, '0' bedeutet, dass i_k invertiert gewählt werden muss (notiert als /i_k), und '-' steht für don't care; d.h., i_k wird nicht verwendet. Werden für yj nur die Werte {0, 1} als Attribute verwendet, spricht man von der ‚kanonisch disjunktiven Normalform'.

Diese Darstellung ist deswegen bedeutend, weil dadurch bei bekannter Reihenfolge der Inputbits die Teilausdrücke zi gemäß vorstehender Gleichung (2), auch als ‚*Terme*' bezeichnet, als sogenannte ‚*Stringterme*' darstellbar sind: Bei einer Reihenfolge i1, i2, i3 bedeutet "111", dass z1 = i1 * i2 * i3 ist, "0-1" steht für z2 = /i1 * i3 usw.

Bei drei Inputbits ist die Menge aller möglichen Inputvektoren v = {000,001,010,011,100,101,110,111}. Falls beispielhaft die Funktion f nur an den Eingangsvektor {001,011,111} den Wert TRUE erhält, brauchen oder dürfen auch nur diese angegeben werden; in Form von Stringterms kann dies durch 111 und 0-1 erfolgen, dies charakterisiert vollständig die gegebene Funktion und ist isomorph zur disjunktiven Normalform
f = /i1 * i3 + i1 * i2 * i3

2.4 Content-Addressable Memory (CAM)

Lese-/Schreib-Speicherbausteine [RAM(Random Addressable Read/Write Memory)] werden üblicherweise zur Speicherung von Daten und Programmen genutzt. In diesem Fall liegt eine Adresse an dem Adressbus an, und nach Ablauf einer Bausteinspezifischen Wartezeit ist beim Lesevorgang das gespeicherte Datum am Datenbus anliegend und kann weiter verwendet werden. Der Schreibvorgang ist in entsprechender Weise arbeitend.

Aus Sicht der Instanz, die die Daten erhalten möchte (z.B. ein Prozessor), ist die Adresse bekannt, und der gespeicherte Inhalt ist vorher unbekannt. Es existieren jedoch Anwendungen, bei der das Verhältnis genau umgekehrt ist: Der gespeicherte Inhalt ist bekannt, und das Interesse ist, an welcher Adresse dieser Inhalt gespeichert ist, wobei die Antwort auch 'nicht vorhanden' sein kann. Speicherbausteine, die diese Art der Abfrage unterstützen, werden als ‚Content-Addressable Memories' [CAMs (Inhalts-adressierbare Speicherbausteine)] bezeichnet.

Speicherbausteine, die als CAM bezeichnet werden und dieses Verhalten direkt unterstützen, gelten als spezielle Bausteine und sind keineswegs häufig anzufinden. Für praktische Anwendungen kann man jedoch die Funktionalität des CAM durch übliche RAM-Bausteine emulieren. Hierzu müssen für alle Daten, die im CAM direkt gespeichert werden würden, bei einer Abfrage jedoch nicht der Wert, sondern die Speicheradresse ergeben, die korrespondierenden Adressen vorher berechnet werden und an der RAM-Adresse, die dem Datum entspricht, gespeichert werden.

2.5 Zellulare Automaten

Zellulare Automaten [CAs (cellular automata)] sind eine Menge von endliche Automaten, die in einem Feld mit feststehender

Topologie angeordnet sind und weitere Eigenschaften besitzen (vgl. Literaturzitate [1] und [4]). Diese Menge von FSMs ist als n-dimensionales Array (meist gilt $n = 2$) angeordnet, wobei jedem Platz feste Koordinaten gegeben sind. Jede FSM besitzt eindeutig Nachbarn, mit denen kommuniziert werden kann. Im Fall $n = 2$ werden meist die 4 umliegenden FSMs (in den 'Himmelsrichtungen' N, E, W, S, daher auch als ‚NEWS-Nachbarschaft' bezeichnet) als Nachbarn angesehen.

Die Kommunikation mit den Nachbarn erfolgt so, dass die Zustände der direkten Nachbarn lesbar und damit auswertbar sind. Mit jedem Zeitschritt wird der Zustand aller Zellen parallel berechnet. Sollen Daten aus weiter entfernten Zellen für die Berechnung eines neuen Zustandes genutzt werden, so müssen diese Daten schrittweise von Zelle zu Zelle transportiert werden. Damit sind klassische zellulare Automaten gut zur Berechnung von Problemstellungen mit hoher Lokalität der Daten geeignet.

CAs gelten als universelle Rechner wie die vorher diskutierten Architekturen; sie arbeiten zudem vollkommen parallel. Soll ein Netz von CAs in eine Hardwareschaltung, z.B. einen ASIC oder auch PLD, abgebildet werden, so steigt die Anzahl der Verbindungen linear mit der Zahl der Automaten an. Die Verbindungen selbst sind je nach gewählter Topologie nur relativ kurz und fest verlegt. Der Aufwand zur Kommunikation der Cas untereinander ist also relativ gering. Wird als Komplexität einer Schaltfunktion der Speicherbedarf angesehen, der nötig ist, diese Schaltfunktion in ein RAM abzubilden, so steigt die maximale Komplexität der dem Verhalten einer Zelle entsprechenden Schaltfunktion exponentiell mit der Anzahl der Eingangsvariablen und linear mit der Anzahl der Ausgangsvariablen der Funktion an. Die Zahl der Eingangsvariablen ist hier die Summe aller Bits, die nötig sind, die Zustände aller mit der Zelle verbundenen CAs einschließlich des Zustandes der Zelle selbst zu codieren. Damit ist die maximale Komplexität der Zelle im Wesentlichen durch die Anzahl der Verbin-

dungen eines jeden Automaten beschränkt.

Das Konzept der globalen zellularen Automaten [GCAs (global cellular automata)] überwindet die Einschränkungen der CAs, indem Verbindungen einer Zelle nicht nur zu ihren nächsten Nachbarn, sondern zu beliebigen Zellen im gesamten Feld erlaubt werden. Damit besitzt ein GCA keine feststehende Topologie mehr sondern ermöglicht, eine an die Problemstellung angepasste und gegebenenfalls zur Laufzeit der Berechnung sogar wechselnde Topologie zu verwenden. Dies kann zu einer erheblichen Beschleunigung in der Programmbearbeitung führen. Die Anzahl der Verbindungen eines einzelnen Automaten ist gegebenenfalls durch eine obere Grenze festgelegt. Ist nur eine einzelne Verbindung erlaubt, so spricht man von einarmigen-, im allgemeinen Fall von k-armigen GCAs.

Als Konsequenz steigt bei der Realisierung eines Feldes von GCAs der erforderliche Aufwand für die Kommunikation der Zellen untereinander mit der Zahl der Zellen stark an. Die Anzahl der möglichen Verbindungen zwischen den einzelnen Automaten steigt quadratisch mit deren Anzahl.

Die Komplexität der einzelnen Automaten selbst bzw. der zugrunde liegenden Schaltfunktion hängt wie bei den konventionellen CAs im Wesentlichen von der Anzahl der Verbindungen einer jeden Zelle ab. Soll ein GCA in eine rekonfigurierbare Schaltung (PLD) abgebildet werden, so muss jede einzelne Zelle, die ja beliebige Schaltfunktionen realisieren kann, die maximal mögliche Komplexität ermöglichen.

Werden die einzelnen Automaten auf jeweils ein Rechenwerk mit lokalem Speicher abgebildet, so kann jede Zelle auch komplexe Schaltfunktionen realisieren. Der Aufwand für eine beliebige Kommunikation alle Zellen steigt quadratisch mit der Anzahl der Zellen. Die Granularität der Schaltung wird dann bestimmt durch die Anzahl der Zellen bzw. die Bitbreite der Verbindungen zwischen den Zellen. Eine solche Schaltung kann sehr gut

GCAs realisieren, die in Anzahl der FSMs und Bitbreite den Vorgaben der Schaltung entsprechen. Es können auch komplexe Schaltfunktionen in jeder einzelnen Zelle realisiert werden. Nachteilig wirkt sich jedoch aus, dass GCAs, die in Anzahl und benötigter Bitbreite der Verbindungen nicht mit der vorgegebenen Körnigkeit übereinstimmen, nur schwierig auf die Schaltung abgebildet werden können.

Werden die einzelnen Zellen als Schaltnetz ausgeführt, so muss jede Zelle in der Lage sein, Daten von allen anderen Zellen einschließlich des eigenen Zustands zu verarbeiten. Aus diesem Grund muss jedes Schaltnetz Schaltfunktionen realisieren können, die alle binärcodierten Zustände aller Zellen als Eingabevariablen enthalten können. Die Anzahl der Ausgabevariablen der Schaltfunktion muss es lediglich ermöglichen, alle Zustände einer einzelnen Zelle binär zu codieren. Nachteilig ist hier, dass die Komplexität der Schaltfunktion exponentiell mit der Anzahl der Eingabevariablen ansteigt. Ebenfalls nachteilig ist der polynomial ansteigende Aufwand für die Kommunikation der Zellen untereinander.

Eine (re-)konfigurierbare Architektur (PLD), die zur Aufnahme eines GCA geeignet ist, muss also pro FSM eine beliebige Komplexität aufnehmen können. Dies bedeutet, dass - falls die FSMs direkt in eine Zelle des PLDs abgebildet werden sollen - die Zellen jede beliebige Funktion aufnehmen müssen. Es ist bekannt, dass sich hieraus ein exponentielles Wachstum der Zellengröße ergibt. Das konfigurierbare Netzwerk in dem PLD muss zudem vollständig ausgeführt sein, d.h., jeder Zellenausgang muss mit jeder anderen Zelle verbindbar sein. Das Netzwerk wächst damit quadratisch an, die Konfiguration des Netzwerks ebenfalls.

Derzeit sind keine PLD-Architekturen am Markt oder bekannt, die beide Forderungen erfüllen: Große Zellen mit einem vollständigen Designraum existieren nicht, es gibt nur kleine Zellen mit vollständigem Designraum [sogenannte ‚Look-Up-

Table-Struktur' (LUT)] oder große Zellen mit unvollständigen Möglichkeiten zur Abbildung beliebiger Funktionen. Eine vollständige Verbindbarkeit aller Zellen ist nur bei großen Zellen möglich. Die Abbildung von GCAs auf existierende PLDs ist damit schwierig, vielfach sogar unmöglich.

3 Aufgabenstellung

Aufgabe der vorliegenden Erfindung ist es, die in der Einleitung 1 angegebene Architektur dahingehend auszugestalten, dass die vorgenannten Probleme zumindest gemindert sind.

4 Lösungsansatz

Die genannte Aufgabe wird erfindungsgemäß mit den in Anspruch 1 angegebenen Maßnahmen gelöst. Demgemäss soll die Architektur mit den eingangs genannten Merkmalen wenigstens ein einzeln konfigurierbares und/oder rekonfigurierbares Schaltnetz aufweisen, dessen Ausgangsvariablen zu einem Zeitpunkt t_{n-1} die Eingangsvariablen des Schaltnetzes zum Zeitpunkt t_n bilden, und mit Mitteln zum Speichern der Ausgangsvariablen des Schaltnetzwerks zwischen den Zeitpunkten t_{n-1} und t_n versehen sein.

Mit den Zeitpunkten t_{n-1} und t_n sind direkt aufeinander folgende Auswertungen der Schaltfunktion bezeichnet. In einer technisch günstigen Realisierung können diese Zeitpunkte von einem eingeprägten Takt mit einer Taktdauer T abgeleitet werden, so dass $t_n - t_{n-1} = k \cdot T$ mit $k=1,2,3,\ldots$ gilt. Für eine gegebene Anwendung ist hierbei k konstant, für verschiedene Anwendungen kann es unterschiedlich gewählt werden.

Nachdem feststeht, dass die Komplexität der Schaltfunktion exponentiell mit der Anzahl der Eingabevariablen und linear mit der Anzahl der Ausgabevariablen bzw. der Anzahl der Zellen im Feld der abzubildenden GCAs steigt und der Aufwand für die Kommunikation der GCAs untereinander zumindest quadra-

tisch mit der Anzahl der Zellen ansteigt, ist die erfindungsgemäße (re-)konfigurierbare Architektur zur Aufnahme von GCAs geeignet. Hierzu besteht sie aus wenigstens einem einzelnen konfigurierbaren Schaltnetz, dessen Ausgangsvariablen zum Zeitpunkt t_{n-1} die Eingangsvariablen des Schaltnetzes zum Zeitpunkt t_n bilden. Zwischen den Zeitpunkten t_{n-1} und t_n werden die Ausgänge dieses Schaltnetzes in einem Speicher wie insbesondere in Registern gespeichert, so dass ein Schaltwerk bzw. eine FSM entsteht.

Vorteilhafte Ausgestaltungen der erfindungsgemäßen Architektur gehen aus den von Anspruch 1 abhängigen Ansprüchen hervor. Dabei kann die Ausführungsform nach Anspruch 1 mit den Merkmalen eines der Unteransprüche oder vorzugsweise auch denen aus mehreren Unteransprüchen kombiniert werden. Demgemäss kann die Architektur noch folgendermaßen ausgebildet sein:

- Als Speichermittel können Registerspeicher vorgesehen sein.
- Bevorzugt ist das Schaltnetz dreistufig ausgebildet.
- Dabei kann vorteilhaft eine erste Stufe mehrere parallelgeschaltete Speicherbausteine enthalten, die über Eingabeleitungen adressierbar sind, wobei jedem Speicherbaustein eine Teilmenge der in einem zugehörenden, ermittelten Implikanten gebundenen Eingabevariablen zuzuführen sind,
  der ersten Stufe eine zweite Stufe mit Speicherbausteinen nachgeordnet sein, die durch Kennungen der einzelnen Implikanten zu adressieren sind,
  und
  der zweiten Stufe eine dritte Stufe mit Mitteln zu einer disjunktiven Verknüpfung der Ausgabewerte der einzelnen Implikanten aus den Speicherbausteinen der zweiten Stufe nachgeordnet sein.
- Dabei lassen sich die einzelnen Implikanten durch Minimierungsverfahren ermitteln.
- Ferner kann die erste Stufe mit der zweiten Stufe über wenigstens einen Crossbar-Switch miteinander verknüpft sein.

- Als Speicherbausteine können vorteilhaft CAMs und/oder RAMs vorgesehen sein.
- Besonders vorteilhaft ist wenigstens ein CGA zu integrieren.
- Als Speicherelemente können magnetoresistive Bauelemente, insbesondere vom TMR-Typ, vorgesehen sein. Entsprechende tunnelmagnetoresistive Elemente sind an sich bekannt.

Den vorstehend angegebenen weiteren Ausgestaltungen der erfindungsgemäßen Architektur liegen insbesondere die nachfolgend dargelegten Überlegungen zu Grunde :

Die Komplexität einer Schaltfunktion in dem gewählten Schaltnetz steigt zwar linear mit der Anzahl der Ausgabevariablen an, jedoch entfällt sämtlicher Aufwand für die Kommunikation der Zellen untereinander. In eine solche Schaltfunktion können viele einzelne Automaten mit wenigen Ausgabevariablen abgebildet werden, oder wenige GCAs mit vielen Ausgabevariablen oder auch eine Mischung verschiedener Zellen. Damit ist keine Granularität vorgegeben, und die Kommunikation der FSMs untereinander ist prinzipiell vollständig möglich. Eine Grenze ist jedoch durch die maximale Komplexität der Schaltfunktion gesetzt, die das Schaltnetz aufnehmen kann.

Die Benutzung eines Schaltnetzes mit einer großen Anzahl von Eingängen - dies wird im allgemeinen Fall entstehen, wenn ein GCA mit einer Anzahl von FSMs abgebildet wird - bedeutet, dass wiederum eine Art exponentielle Abhängigkeit des Flächenbedarfs von der Anzahl der Eingänge entstehen kann. Als obere Grenze gilt ein Wachstum der Fläche mit exp(Anzahl Zustände), wenn jeder Zustand in einem Bit codiert wird; im allgemeinen Fall wird der Flächenbedarf geringer sein. Da eine universelle Schaltung jedoch den Maximalfall beinhalten muss, wäre das exponentielle Wachstumsgesetz anzuwenden.

Hier ist es als besonders vorteilhaft anzusehen, die Schaltung, die das Schaltnetz aufnimmt, in drei spezielle Ab-

schnitte/Stufen einzuteilen. Dazu wird nachstehend eine (re)konfigurierbare Schaltung dargestellt, die Schaltfunktionen mit einer großen Zahl von Eingangsvariablen und einer großen Zahl von Ausgangsvariablen als Schaltnetz realisieren kann.

Um eine rekonfigurierbares Schaltnetz für Schaltfunktionen zu entwerfen, werden als Ausgangsüberlegung zwei Möglichkeiten betrachtet:

Zum einen ist es möglich, eine Schaltfunktion komplett in einem RAM-Speicher abzulegen. Die Eingabevariablen der Schaltfunktion bilden die Adressbits und adressieren für jede mögliche Kombination von Eingabevariablen eine Speicherzelle. Der Inhalt dieser Speicherzelle entspricht dann dem Wert der Schaltfunktion, die Datenleitungen des Speicherbausteins bilden die Ausgabevariablen der Funktion. Der Vorteil dieses Konzepts liegt im einfachen Aufbau der Schaltung, der einfachen Rekonfigurierbarkeit, der hohen Integrationsdichte von Speicherbausteinen und der festen Zeitdauer, die die Auswertung der Schaltfunktion benötigt. Allerdings steigt die Anzahl der benötigten Speicherzellen, also die Größe des benötigten Speicherbausteines exponentiell mit der Anzahl der Eingabevariablen an. Aus diesem Grunde können nur kleine Schaltfunktionen auf diese Weise abgebildet werden.

Dies ist Stand der Technik in PLDs und wird als Look-Up-Table-Struktur bei FPGAs eingesetzt, meist mit 4 binärwertigen Eingangsvariablen und 1 binärwertigen Ausgang.

Eine zweite Möglichkeit, Schaltfunktionen in ein Schaltnetz abzubilden, besteht darin, Gatter in einem 2- oder mehrstufigen Netz konfigurierbar anzuordnen. Damit ist es möglich, Schaltfunktion mit einem minimalen Verbrauch von Gattern in Schaltnetze abzubilden. Die Schaltkreistheorie stellt hier günstige Darstellungsformen von Schaltfunkti-

onen wie z.B. die Reed-Muller-Form, oder auch leistungsfähige Algorithmen zur Logikminimierung bereit. Der Vorteil dieses Ansatzes besteht im minimalen Verbrauch von Gattern und in der Möglichkeit, leistungsfähige Verfahren und Algorithmen zur Minimierung zu nutzen (vgl. Literaturzitate [2] und [3]). Dieses Verfahren kann gut zur Darstellung einer festen Schaltfunktion z.B. der Realisierung eines ROMs genutzt werden. Über Hardwarebausteine wie z.B. Crossbar-Switches können die einzelnen Gatter rekonfigurierbar verschaltet werden, jedoch steigt hier der Aufwand für die Rekonfigurierbarkeit exponentiell mit der Anzahl der Eingabevariablen der Schaltfunktion an.

5 Erläuterung der Erfindung anhand eines konkreten Ausführungsbeispiels

Um die Vorteile der ersten Möglichkeit, die hohe Integrierbarkeit, die Rekonfigurierbarkeit und die Einfachheit der Schaltung, und die Vorteile der 2. Variante, den geringen Verbrauch von Gatterfunktionen und die Anwendbarkeit moderner Verfahren, verbinden zu können, ist erfindungsgemäß folgender Ansatz vorgesehen:

Die Grundidee der angenommenen Ausführungsform ist die Entwicklung eines rekonfigurierbaren Schaltnetzes, welches soviel logische Funktionalität wie möglich in RAM-Bausteinen darstellt, um den Vorteil der hohen Integrationsdichte zu nutzen. Diese RAM-Bausteine sollen in einer mehrstufigen Schaltung rekonfigurierbar miteinander verbunden werden, um nicht komplette Schaltfunktionen speichern zu müssen.

1. Stufe:
Die Eingabevariablen der Schaltfunktion werden durch die Eingabeleitungen des Schaltnetzes repräsentiert. Die erste Stufe der Schaltung besteht aus mehreren parallel geschalteten Speicherbausteinen, die durch die Eingabeleitungen adressiert werden. Jedem Speicherbaustein ist dabei eine Teilmenge der

Eingabeleitungen und damit der Eingabevariablen der Schaltfunktion zugeordnet. Die durch Minimierungsverfahren ermittelten Implikanten einer Schaltfunktion (minimierte zi aus Gl. (2) bzw. deren Stringterm-Darstellung) werden in Speicherbausteinen der Eingangsstufe abgespeichert. Dazu werden in jedem Speicherbaustein, dem im Implikanten gebundene Eingabevariablen zugeordnet sind, die Belegung der Eingabevariablen im Implikanten und eine eindeutige Kennung zu diesem Implikanten abgespeichert. Da jedem Baustein nur ein Teil der Eingabevariablen zugeordnet sind, wird auch nur jeweils ein Teilimplikant gespeichert. In Speicherbausteinen, denen keine im Implikanten gebundenen Variablen zugeordnet sind, wird entsprechend kein Teil des Implikanten gespeichert. Liegt an den Eingabeleitungen des Schaltnetzes eine Bitkombination an, so geben alle Speicherbausteine, die zur Bitkombination passende Teilimplikanten enthalten, deren
Kennungen über die Datenleitungen an die 2. Stufe der Schaltung weiter.

2. Stufe:
Die Kennungen der (Teil-)Implikanten adressieren einen Speicher in einer zweiten Stufe. In diesem Speicher sind die zum jeweiligen Implikanten gehörigen Bitmuster der Kennungen und die Ausgabewerte der Schaltfunktion gespeichert. Entspricht das Bitmuster an Kennungen, die von der ersten Stufe geliefert werden, dem eines gespeicherten Implikanten, so liegt dieser Implikant an den Eingangsleitungen der Schaltung an. Die 2. Stufe der Schaltung leitet dann die Ausgabewerte aller Implikanten, die an den Eingabeleitungen anliegen, über die Datenleitungen an die 3. Stufe weiter.

3. Stufe:
In der 3. Stufe werden die Ausgabewerte der einzelnen Implikanten disjunktiv (OR) verknüpft und bilden so das Ergebnis der Schaltfunktion.

Die Erfindung wird nachfolgend unter Bezugnahme auf die Zeichnung anhand eines konkreten Ausführungsbeispieles unter Berücksichtigung der vorstehenden Überlegungen noch weiter erläutert. Die Zeichnung umfasst folgende Teile, wobei deren Figuren 1 bis 3 zum Stand der Technik unter Abschnitt 2 bereits angesprochen wurden :

Figur 1 zeigt den prinzipiellen Aufbau einer Von-Neumann-Architektur gemäß dem Stand der Technik,

Figur 2 zeigt die generelle Struktur eines PLD gemäß dem Stand der Technik,

Figur 3 zeigt den prinzipiellen Aufbau einer FSM in Form eines Mealy-Automaten gemäß dem Stand der Technik,

Figur 4 zeigt ein Ausführungsbeispiel eines Speicheraufbaus einer erfindungsgemäßen Architektur,

Figur 5 zeigt die Abbildung von Springtermen auf RAM, wobei Teilfigur a) partielle Springterme, Teilfigur b) die Abbildung auf ein Tag-RAM und Teilfigur c) die Abbildung auf ein konventionelles RAM veranschaulichen,

Figur 6 zeigt die Abbildung des Ergebnisses der Stufe 1 einer erfindungsgemäßen Architektur auf eine RAM-Kombination in Stufe 2,

Figur 7 zeigt eine endgültige Architektur für das Beispiel $[(0,1)^{12} \rightarrow (0,1)^{12}$-Funktion],

Figur 8 zeigt eine erfindungsgemäße Architektur für ein Schaltwerk mit großem Schaltnetz zur Aufnahme eines GCA

und

Figur 9 eine erfindungsgemäße rekonfigurierbare Architektur zur Aufnahme von CGAs.

Dabei sind in den Figuren sich entsprechende Teile jeweils mit denselben Bezugszeichen versehen.

Für das Ausführungsbeispiel gemäß den Figuren 4 bis 9 sei eine erfindungsgemäße Architektur mit drei Stufen, wie vorstehend angesprochen, angesetzt, und zwar für eine Schaltfunkti-

on mit 12 Eingabevariablen, 10 Implikanten und 8 Ausgabevariablen: Tabelle 1 zeigt hierfür alle Implikanten (auch als ‚Minterme' bezeichnet) für eine Beispielfunktion an. Die Darstellung der Stringterme ist so gewählt, dass hierbei drei Vierergruppen entstehen.

Tabelle 1: Beispiel für eine Implikantentabelle, dargestellt durch Stringterme

Implikanten:

1.	0-11 ---- 1100		6.	0100 11-- 0000	
2.	1101 11-- 0000		7.	---- 0001 0000	
3.	-0-1 0001 ----		8.	---- 0001 ----	
4.	---- ---- --10		9.	---- ---- 0000	
5.	0100 0001 ----		10.	-0-1 0001 1100	

Betrachtet man nun diese Tabelle spaltenweise, wird man feststellen, dass nur wenige verschiedene Kombinationen in den Stringtermen vorkommen. Bei einer zweiwertigen Darstellung könnte es für jede Spalte hier 24 = 16 verschiedene Kombinationen geben, bei dreiwertiger entsprechend 34 = 81. In dem Beispiel kommen hiervon nur 5, 3 und 4 für die Spalten 1 – 3 vor, wobei eine Eingangskombination jeweils komplett '-' ist.

Liegt am Eingang eine Bitkombination als Parameter der Schaltfunktion an, so liefert ein Speicherbaustein, welcher einen Teilimplikanten mit der anliegenden Bitkombination speichert, die Kennung des zugehörigen Implikanten zurück. Dieser Speicherbaustein der ersten Stufe ist in Figur 4 als 3-wertiges CAM ausgeführt, d.h., die Eingangsvektoren, die real als zweiwertige Information an dem Adressbus anliegt, wird mit gespeicherter dreiwertigen Informationen verglichen. Als Ausgabe wird eine zu dem Treffer gespeicherte Kennung, ebenfalls dreiwertig, ausgegeben.

Alle Kennungen zusammen bilden die Kennung der Implikantenkombination, die am Eingang des Schaltnetzes anliegt. Diese Implikantenkombination kann dabei durchaus mehrere Implikan-

ten umfassen. So können im dargestellten Beispiel z.B. die Implikanten 3, 4 und 8 oder die Kombination der Implikanten 4, 5 und 8 anliegen. Im ersten Fall liegt die Bitkombination 100001 an der 2. Stufe an, im zweiten Fall die Kombination 110001.

Die Kennung der Implikantenkombinationen wird in der 2. Stufe der Schaltung erkannt und liefert für jeden beteiligten Implikanten den zugehörigen Ausgabewert der Schaltfunktion. Diese zweite Stufe besteht nun aus einem dreiwertigen RAM, d.h., am die Adressbusinformationen dieser Stufe sind dreiwertig, die gespeicherten Daten allerdings zweiwertig.

In der 3. Stufe der Schaltung werden die Ausgabewerte der anliegenden Implikanten disjunktiv verknüpft und bilden zusammen den Funktionswert der Schaltfunktion.

5.1 Speicherbausteine Stufe 1

Als Speicherbaustein zum Aufnehmen der Teilimplikanten kann - wie schon erwähnt - ein sehr spezieller Baustein bzw. Architektur zum Einsatz kommen, hier mit dreiwertigem CAM gemäß Figur 4 bezeichnet. Hierbei muss prinzipiell noch das Problem der Mehrfachübereinstimmung diskutiert werden, was aber im Zusammenhang mit anderen Realisierungsmöglichkeiten erfolgen soll.

Möglich als Einsatz für die Stufe 1 ist auch ein vollassoziativer Cache. Hier können die Teilimplikanten als sogenannter Tag gespeichert werden, das gecachte Datum dient als Kennung des erkannten Implikanten. Enthält jedoch ein Teilimplikant ungebundene Variablen, die beim Vergleich mit anliegenden Bitkombinationen als Don't-Care(DC)-Stellen zum Ausdruck kommen, so muss für alle Belegungen dieses Implikanten, die den Vergleich mit DC erfüllen, ein Tag im Tag-RAM angelegt werden. Weiterhin ergeben sich durch den Vergleich mit DC Überschneidungen von Teilimplikanten. So gehört z.B. die Bit-

kombination 0011 im ersten Teilimplikantenspeicher des obigen Beispiels sowohl zum Implikanten 1 als auch zum Implikanten 3. Es sind also nicht nur Kombinationen von Implikanten möglich, sondern auch Kombinationen von Teilimplikanten.

Aus diesem Grunde - und dies gilt als der Vorschlag zur Realisierung - werden als Teilimplikantenspeicher normale, d.h. zweiwertige RAM-Bausteine bzw. -Architekturen verwendet. Jedes dieser RAMs wird durch einen Teil der Eingangsleitungen des Schaltnetzes adressiert. An den Adressen, deren Bitkombination jeweils demselben Teilimplikanten mit DC-Stellen entsprechen (jede DC-Stelle in einem Stringterm bedeutet, dass die Anzahl der zutreffenden Stellen bei binärer Codierung um den Faktor 2 erhöht wird), wird jeweils die gleiche Kennung im Speicher abgelegt. Ist die Ausgangsbreite des verwendeten RAMs größer als die zur Darstellung der Kennungen notwendige Bitbreite, so können die weiteren Bits als Kontextwert genutzt werden. Dieser Kontext kann z.B. eine ungültige Belegung der Eingangsvariablen der Schaltfunktion anzeigen.

Da ein normales RAM keine Statusanzeige für ein nicht vorhandenes Datum wie z.B. ein Tag-RAM mit seinem Cache-Miss-Ausgang besitzt, muss eine Bitkombination auf denjenigen Datenleitungen, die an die 2. Stufe zum Vergleich der Bitkombinationen verwendet werden, als Kennzeichnung für keinen anliegenden Teilimplikanten verwendet werden. In Figur 5 c) ist dies durch die Kennung 8 gegeben.

5.2 Speicherbausteine Stufe 2

Ebenso wie im bei den RAMs der ersten Stufe zum Speichern der Teilimplikanten müssen auch beim Vergleich der Implikantenkombinationen in der 2. Stufe der Schaltung DCs berücksichtigt werden. Deshalb wird auch hier ein normales RAM verwendet. Dieses RAM wird mit der Kennung der Implikantenkombination adressiert.

Da wieder mehrere Adressen derselben Implikantenkombination entsprechen können, muss der Speicher der zweiten Stufe aufgeteilt werden: die Bitkombinationen der ersten Stufe adressieren ein RAM der zweiten Stufe. Dort ist für jede gültige Implikantenkombination ein Index abgelegt, der wiederum ein RAM adressiert, welches seinerseits die Ausgabevariablen der beteiligten Implikanten enthält. So können die verschiedenen Adressen, die sich durch die Implikantenkombination mit Don't-Care-Stellen ergeben, auf denselben Index der Ausgabewerte der Schaltfunktion abgebildet werden.

Figur 6 zeigt eine entsprechende Abbildung des Ergebnisses der Stufe 1 auf eine RAM-Kombination in Stufe 2.

5.3 Endgültige Architektur gemäß Figur 7

Da ein RAM nur einen einzigen Index liefern kann, müssen im Speicher der Ausgabewerte die disjunktiv verknüpften Ausgabewerte aller an der erkannten Kombination beteiligten Implikanten werden. Damit muss das Ausgabe-RAM alle möglichen Funktionswerte der Schaltfunktion speichern können. Da die Anzahl der möglichen Funktionswerte exponentiell mit der Anzahl der Ausgabevariablen einer Schaltfunktion ansteigt, werden mehrere Kombinations- und Ausgabe-Speicher der 2. Stufe parallel verwendet und deren Ausgabekombinationen disjunktiv verknüpft. Damit können alle Funktionswerte einer Schaltfunktion erzeugt werden.

Um die Kapazität der einzelnen Kombinationsspeicher besser ausnutzen zu können, werden alle Datenleitungen der Implikantenspeicher mit allen Adressleitungen der Kombinationsspeicher über einen Crossbar-Switch verbunden. Damit können beliebige Datenleitungen die Adressierung der Kombinations-RAMs übernehmen. Nicht verwendete Datenleitungen können über den Crossbar-Switch als Kontext-Information weitergeleitet werden.

In einem letzten Schritt wird der erzeugte Ausgabewert bitweise über die Exklusiv-Oder-Funktion mit einem Registerinhalt verknüpft, um einzelne Ausgabevariablen invertieren zu können und so ggf. kleinere Logikminimierungen zu erhalten. Damit besteht die 3. Stufe der Schaltung aus der disjunktiven Verknüpfung der Ausgabekombinationen und der anschließenden möglichen Invertierung einzelner Ausgabebits.

Insgesamt ergibt sich das Prinzipschaltbild der Figur 7 für ein erfindungsgemäßes Schaltnetz.

5.4 Diskussion der Architektur

Die Intention der Erfindung ist es, sowohl eine RAM-basierte Architektur zur Implementierung großer Schaltnetze als auch - gewissermaßen als Anwendung zur Aufnahme einer universellen Maschine - diese Architektur zur Aufnahme von GCAs anzubieten. Zur Aufnahme einer beliebigen Funktion im RAM muss der Speicherplatz in dem Speicher exponentiell mit der Anzahl der Eingänge (und linear mit der Anzahl der Ausgänge) wachsen. Im Fall des obigen Beispiels bedeutet dies, dass eine beliebige Funktion mit 12 Ein- und 12 Ausgängen einen Speicherbedarf von 4096 * 12 bit entsprechend 6144 Bytes hätte. Bei 32 Eingängen und 8 Ausgängen wären dies bereits 4 GByte an Speicherkapazität.

Die vorgeschlagene Architektur eines mehrstufigen Netzwerks beinhaltet lediglich 211,5 Bytes RAM, nämlich:

3x Implikanten-RAM 16x4	24 Bytes
3x Kombinations-RAM 64x4	96 Bytes
3x Ausgabe-RAM 16x12	72 Bytes
Crossbar-Switch-Konfiguration	18 Bytes
Invertierung 12x1	1,5 Bytes
Summe	211,5 Bytes

Damit liegt der wesentliche Vorteil darin, dass diese Archi-

tektur erheblich platzsparender ist als eine LUT-basierte Architektur. Hierbei ist zu berücksichtigen, dass nicht jede Funktion auf diese Weise darstellbar ist.

Um eine Applikation in diese Architektur abbilden zu können, müssen mehrere notwendige Bedingungen erfüllt sein. Die erste Bedingung ist diejenige, dass die Anzahl der verschiedenen Teil-Stringterme, die in einer Spalte vorhanden sind, auf die RAMs der ersten Stufe abgebildet werden können. Dies ist automatisch dadurch erfüllt, dass diese RAMs alle Kombinationen aufnehmen (weil sie CAMs emulieren), lediglich die Eingangsbreite der Schaltung muss für Applikation ausreichen.

Die zweite Bedingung schließt sich hier an: Die Anzahl der verschiedenen Teil-Stringtermkombinationen, die in der Applikation nach Minimierung enthalten sind, muss codierbar sein. Dies bedeutet, dass eine Anzahl von Speicherstellen zur Verfügung stehen muss. Zur Effizienzabschätzung sei m die Eingangsbreite des Schaltnetzes. Dies würde bedeuten, dass 2m Speicherzellen benötigt würden, um die komplette Funktion darzustellen.

Wenn k nun die Parallelität der Bausteine (Anzahl der Bausteine) und 2s die Kapazität eines Bausteins ist, so muss für eine effizientere Speicherung der Applikation die Ungleichung

$$k * s \leq m-1 \qquad (3)$$

gelten. Je deutlicher die Unterschreitung ausfällt, desto effizienter war die Implementierung.

Bedingung 3 bedeutet, dass die Ausgangsbreite geeignet gewählt sein muss.

6 Abbildung von GCAs auf die Architektur

Zur Abbildung von GCAs auf die erfindungsgemäße Architektur müssen noch Speicherelemente eingeführt werden, die taktgesteuert das Fortschreiten in der Rechnung speichern. Dies

hat seine Ursache darin, dass GCAs als Array von FSMs definiert sind, und diese sind in der Regel synchronisiert. Hier ist angenommen, dass ein globaler Takt zur Synchronisation genommen wird. Alle Implementierungen von nicht-globalen, insbesondere nicht in gegenseitiger Beziehung stehenden Takten würden zu wesentlichen Problemen führen, sind jedoch in der Praxis selten anzutreffen.

Figur 8 zeigt ein weiteres Beispiel für eine konfigurierbare Architektur der vorgeschlagenen Art, nunmehr ausgestattet mit Registern zur Speicherung von Zuständen. Zusätzlich ist ein weiterer Crossbar-Switch eingefügt, der an dieser Stelle u.a. dazu dient, Ein- und Ausgabeschnittstellen für den Rechner bereitzustellen. Dies ist zwar für das grundlegende Verständnis der Architektur unerheblich, im praktischen Betrieb jedoch notwendig, weil ein Rechner mit Außenanschlüssen versehen sein muss.

Der Speicherbedarf dieser beispielhaften Architektur berechnet sich zu

8x minterm-RAM 256x8	2 KBytes
8x combination-RAM 64Kx8	512 KBytes
8x output-vector-RAM 256x64	16 KBytes
2xCrossbar-Switch configuration	1 KBytes
Inverting register 64x1	8 Bytes
Sum	**531 KBytes**

Damit ist verdeutlich, wie gering die Speicheranforderungen sind; ein RAM mit 264 * 64 bit (=267 Bytes) ist jedenfalls nicht in der Herstellung möglich. Sollten bei einer Applikation auf dieser Architektur Leitungen von den RAMs der ersten Stufe ungenutzt bleiben, können diese als Kontextinformationen genutzt werden. Eine Anwendung besteht dabei in der Kontextumschaltung, die für die RAMs der zweiten Stufe zusätzlich möglich sein könnte. Werden also beispielsweise bei einem RAM nur 14 Adressbits benötigt, dann können die Informa-

tionen für diese 14 bit viermal gespeichert werden, also in vier verschiedenen Kontexten stehen.

Aus Figur 9 ist der prinzipielle Aufbau einer rekonfigurierbaren Architektur nach der Erfindung zu entnehmen, wie sie zur Aufnahme von CGAs geeignet ist. Dieser Aufbau stellt eine Verallgemeinerung des Aufbaus nach Figur 8 dar. Insbesondere sind die RAM-Stufen 1 und 2 durch gestrichelte Linien verdeutlicht.

7 Literaturzitate

[1] Rolf Hoffmann, Klaus-Peter Völkmann, Wolfgang Heenes: "Globaler Zellularautomat (GCA): Ein neues massivparalleles Berechnungsmodell", Mitteilungen - Gesellschaft für Informatik e.V., Parallel-Algorithmen und Rechnerstrukturen, ISSN 0177-0454 Nr. 18, 2001, Seiten 21-28; http://www.ra.informatik.tu-darmstadt.de/publikationen/publik.html

[2] R.K.Brayton et.al.: "Logic Minimization Algorithms for VLSI Synthesis", Kluwer Academic Publishers, USA 1984.

[3] Mike Trapp: "PLD-design methods migrate existing designs to high-capacity devices", EDN Access, Febr. 1994; http://www.reed-electronics.com/ednmag/archives/1994/021794/04df1.htm

[4] Wolfgang Heenes, Rolf Hoffmann, Klaus-Peter Völkmann: "Architekturen für den globalen Zellularautomaten".19th PARS Workshop, March 19-21, 2003 Basel; http://www.ra.informatik.tu-darmstadt.de/publikationen/pars03.pdf

Patentansprüche

1. Rekonfigurierbare Architektur einer Rechnereinrichtung mit wenigstens einem einzeln konfigurierbaren und/oder rekonfigurierbaren Schaltnetz, dessen Ausgangsvariablen zu einem Zeitpunkt t_{n-1} die Eingangsvariablen des Schaltnetzes zu einem Zeitpunkt t_n bilden, und mit Mitteln zu einem taktgesteuerten Speichern der Ausgangsvariablen des Schaltnetzes zwischen den Zeitpunkten t_{n-1} und t_n.

2. Architektur nach Anspruch 1, dadurch gekennzeichnet, dass die Speichermittel Registerspeicher sind.

3. Architektur nach Anspruch 1 oder 2, gekennzeichnet durch eine dreistufige Ausbildung des Schaltnetzes.

4. Architektur nach Anspruch 3, gekennzeichnet
- durch eine erste Stufe aus mehreren parallel geschalteten Speicherbausteinen, die über Eingabeleitungen adressierbar sind, wobei jedem Speicherbaustein eine Teilmenge der in einem zugehörenden, ermittelten Implikanten gebundenen Eingabevariablen zuzuführen sind,
- durch eine der ersten Stufe nachgeordnete zweite Stufe mit Speicherbausteinen, die durch die Kennungen der einzelnen Implikanten zu adressieren sind,
  und
- durch eine der zweiten Stufe nachgeordnete dritte Stufe mit Mitteln zu einer disjuktiven Verknüpfung der Ausgabewerte der einzelnen Implikanten aus den Speicherbausteinen der zweiten Stufe.

5. Architektur nach Anspruch 4, gekennzeichnet durch eine Ermittlung der Implikanten durch Minimierungsverfahren.

6. Architektur nach Anspruch 4 oder 5, dadurch gekennzeichnet, dass die erste Stufe mit der zweiten Stufe über wenigstens einen Crossbar-Switch miteinander verknüpft ist.

7. Architektur nach einem der vorangehenden Ansprüche, gekennzeichnet durch CAMs- und/oder RAMs als Speicherbausteine.

8. Architektur nach einem der vorangehenden Ansprüche, gekennzeichnet durch eine Integration wenigstens eines CGAs.

9. Architektur nach einem der vorangehenden Ansprüche, gekennzeichnet durch magnetoresistive Speicherelemente, insbesondere vom TMR-Typ.

Zusammenfassung

Rekonfigurierbare Architektur zur parallelen Berechnung beliebiger Algorithmen

Die konfigurierbare Architektur einer Rechnereinrichtung weist wenigstens ein einzeln konfigurierbares und/oder rekonfigurierbares Schaltnetz auf, dessen Ausgangsvariablen zu einem Zeitpunkt t_{n-1} die Eingangsvariablen des Schaltnetzes zu einem Zeitpunkt t_n bilden. Es sind Mittel zu einem taktgesteuerten Speichern der Ausgangsvariablen des Schaltnetzes zwischen den Zeitpunkten t_{n-1} und t_n vorgesehen.

FIG 9

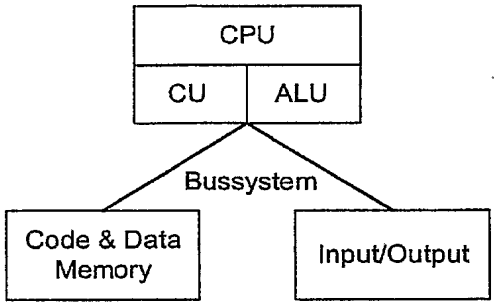
CPU
CU
ALU
Bussystem
Code & Data Memory
Input/Output

FIG 1

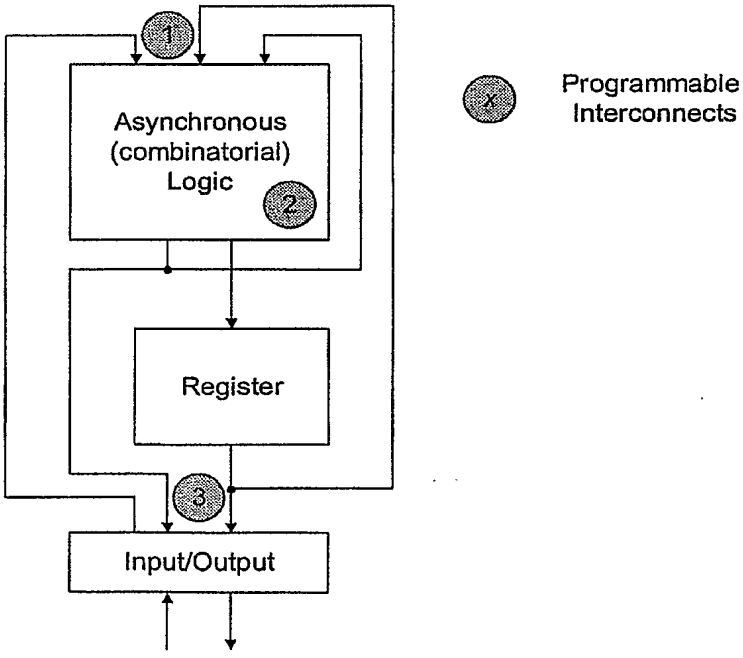
1
Asynchronous (combinatorial) Logic
2
Register
3
Input/Output
x
Programmable Interconnects

FIG 2

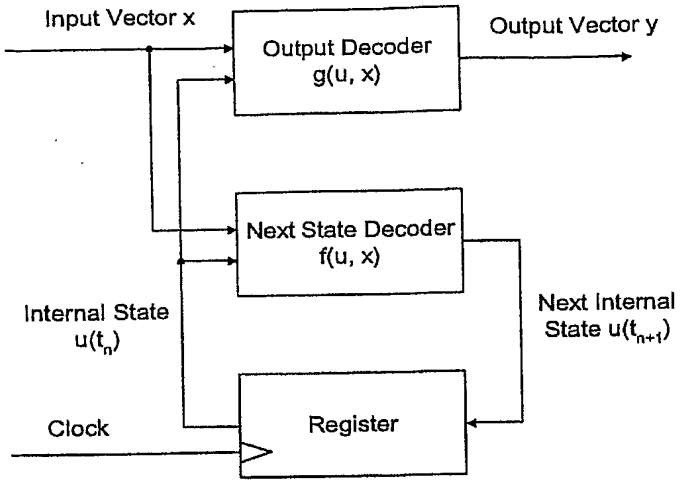
Input Vector x
Output Decoder
g(u, x)
Output Vector y
Next State Decoder
f(u, x)
Internal State
$u(t_n)$
Next Internal
State $u(t_{n+1})$
Clock
Register

FIG 3

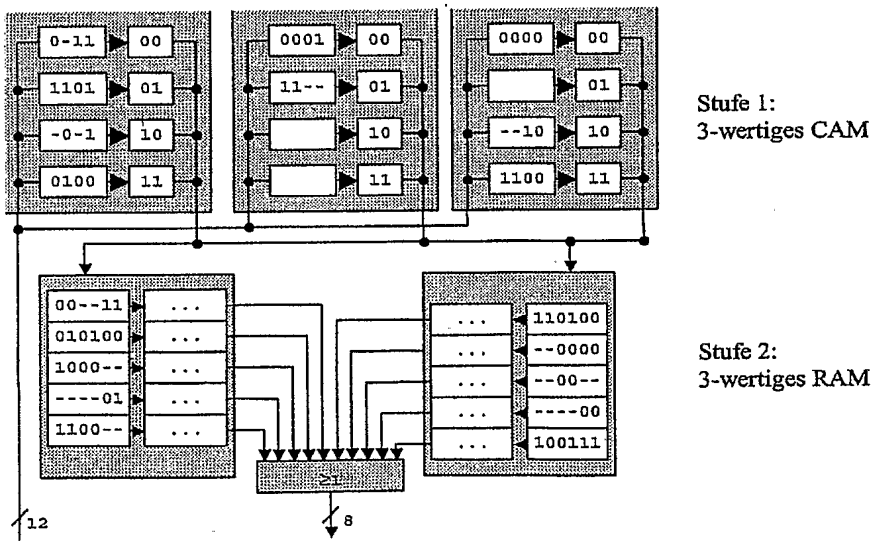
0-11
00
1101
01
-0-1
10
0100
11
0001
00
11--
01
10
11
0000
00
01
--10
10
1100
11
00--11
010100
1000--
----01
1100--
...
110100
--0000
--00--
----00
100111
12
8
Stufe 1:
3-wertiges CAM
Stufe 2:
3-wertiges RAM

FIG 4

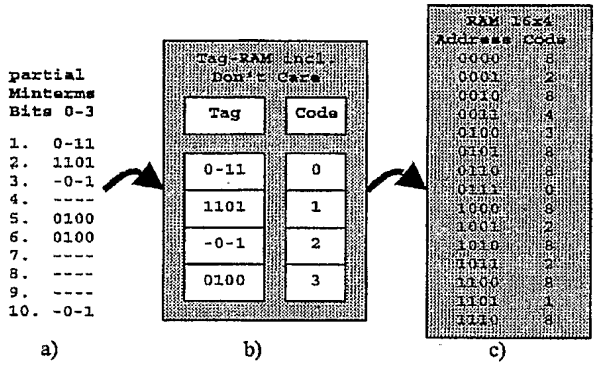
partial
Minterms
Bits 0-3
1. 0-11
2. 1101
3. -0-1
4. ----
5. 0100
6. 0100
7. ----
8. ----
9. ----
10. -0-1
Tag-RAM incl. Don't Care
Tag
Code
0-11
0
1101
1
-0-1
2
0100
3
RAM 16x4
Address Code
0000 3
0001 2
0010 8
0011 4
0100 3
0101 8
0110 8
0111 C
1000 8
1001 2
1010 8
1011 2
1100 8
1101 1
1110 8
a)
b)
c)

FIG 5

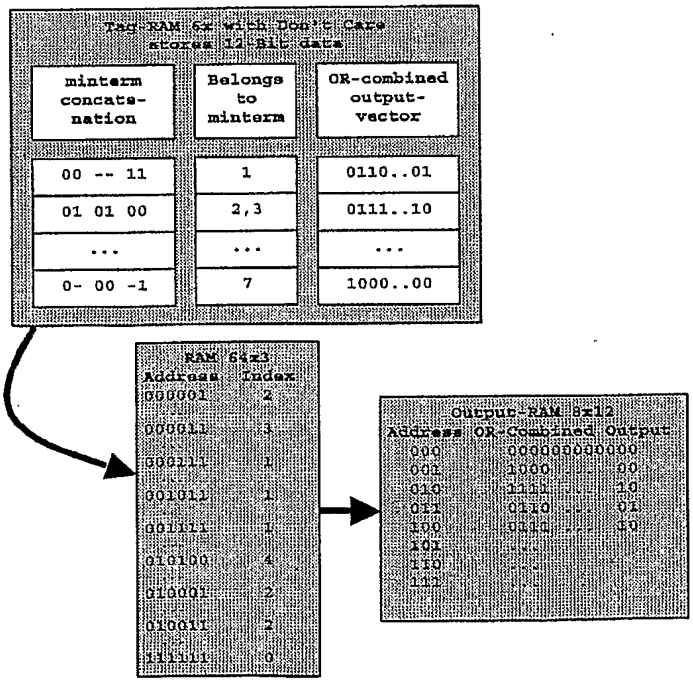
Tag-RAM 6x with Don't Care
stores 12 Bit data
minterm concate-nation
Belongs to minterm
OR-combined output-vector
00 -- 11
1
0110..01
01 01 00
2,3
0111..10
...
...
...
0- 00 -1
7
1000..00
RAM 64x3
Address Index
000001 2
000011 3
000111 1
001011 1
001111 1
010100 4
010001 2
010011 2
111111 0
Output-RAM 8x12
Address OR-Combined Output
000 000000000000
001 1000 .. 00
010 1111 .. 10
011 0110 .. 01
100 0111 .. 10
101 ...
110 ...
111 ...

FIG 6

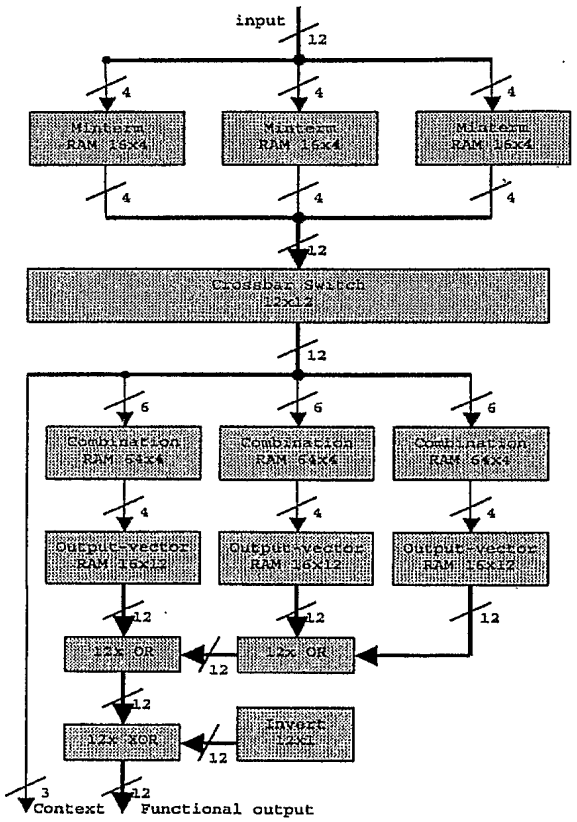
input
12
4
4
4
Minterm RAM 16x4
Minterm RAM 16x4
Minterm RAM 16x4
4
4
4
12
Crossbar Switch 12x12
12
6
6
6
Combination RAM 64x4
Combination RAM 64x4
Combination RAM 64x4
4
4
4
Output-vector RAM 16x12
Output-vector RAM 16x12
Output-vector RAM 16x12
12
12
12
12x OR
12
2x OR
12
12
12x XOR
12
Invert 12C
3
12
Context
Functional output
Stage 1
Stage 2
Stage 3

FIG 7

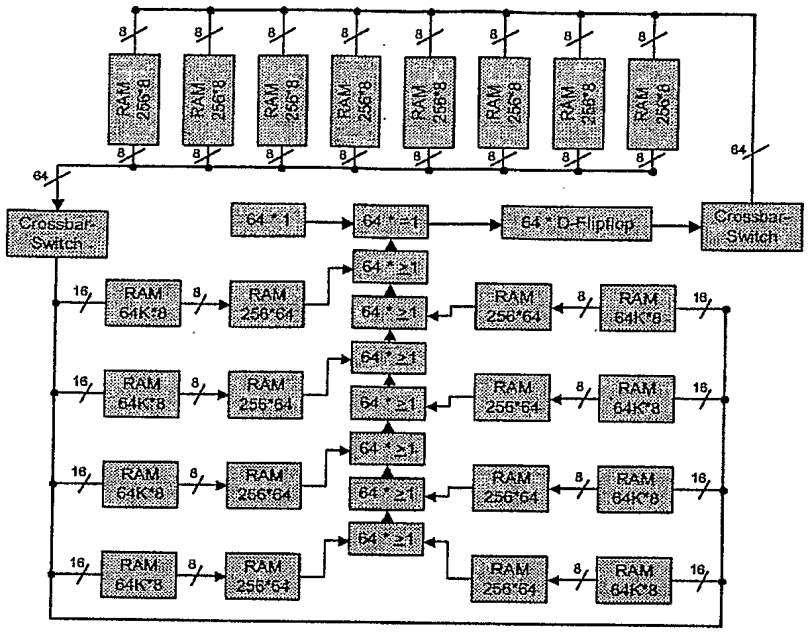
RAM 256*8
Crossbar-Switch
64 * 1
64 * ≥1
64 * D-Flipflop
RAM 64K*8
RAM 256*64

FIG 8

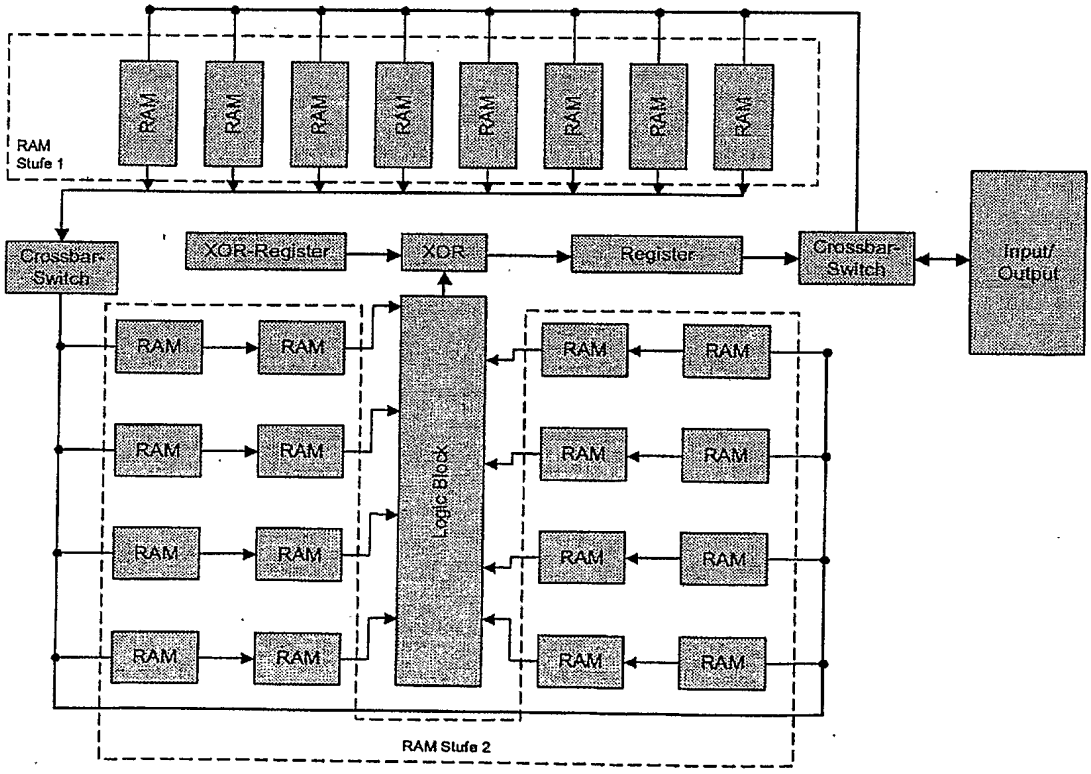
RAM
RAM Stufe 1
Crossbar-Switch
XOR-Register
XOR
Register
Crossbar-Switch
Input/ Output
Logic Block
RAM Stufe 2

FIG 9